Analysis of the use of complete orders to abstract the internet connectivity at the autonomous system level

Arjona-Villicana, Pedro; Constantinou, Costas; Stepanenko, Alexander; Jesús Acosta-Elías, J.

DOI:
10.1049/iet-net.2017.0020

IET Research Journals

The Institution of Engineering and Technology

An Analysis of the Use of Complete Orders to Abstract the Internet Connectivity at the Autonomous System Level

ISSN 1751-8644
doi: 0000000000
www.ietdl.org

P. David Arjona-Villicaña[1*] *Costas C. Constantinou*[2] *Alexander S. Stepanenko*[3] *J. Jesús Acosta-Elías*[4]

[1] *Facultad de Ingeniería, Universidad Autónoma de San Luis Potosí, San Luis Potosí, Mexico*
[2] *Department of Electronic, Electrical and Computer Engineering, University of Birmingham, Edgbaston, Birmingham, B15 2TT, United Kingdom*
[3] *Fusion Asset Management LLP, London, SW1P 3AE, United Kingdom*
[4] *Facultad de Ciencias, Universidad Autónoma de San Luis Potosí, San Luis Potosí, Mexico*
** E-mail: darjona@yahoo.com*

Abstract: Complete orders, also called chains, are transitive acyclic digraphs which can be employed as a topological unit to model, abstract and exploit the Internet's path diversity. The main objective behind this work is to demonstrate why complete orders are well-suited in abstracting routing information at the interdomain level. In order to abstract a network's topological information, it becomes necessary to introduce another mathematical structure called Virtual Arc, which allows to define complete orders where these cannot be formed directly. An algorithm, called Chain Routing, that employs Virtual Arcs to define chains in a network is described and tested. Further analysis of the Chain Routing algorithm leads to the conclusion that this is an NP-Complete problem; however, by limiting the size of the complete orders to be used, it is possible to provide an upper bound to the computational task needed to discover these structures in the Internet.

1 Introduction

The Border Gateway Protocol (BGP) [1] is the *de facto* routing protocol used to transmit information through the networks or Autonomous System (AS) that form the Internet. BGP is implemented as a decentralised routing algorithm that at a high-level works in the following way: every BGP router independently computes and selects a preferred path to each destination in its routing table and then announces these paths to its neighbouring routers. As new routers receive these announcements, they also select a best path to a destination and announce the new path to its corresponding neighbours. As these destination announcements propagate through the Internet, ASs learn where to send information back for each destination. When all the routers in a network adopt a stable set of paths for a destination, it is said that the network *converges* or *finds a solution*.

Research in BGP's performance has uncovered two problems with this routing protocol. The first one is the *convergence delays* BGP suffers when trying to find a solution for its paths [2–4], caused by the excessive number of messages BGP needs to propagate changes in the network. Strategies to solve this problem [5, 6] implement a complete order in the realm of time [7]. However, this problem is not addressed in this paper.

By default, BGP selects the shortest path to each destination in its routing table, however this mechanism is usually customised to suit the needs of the AS's network administrator. This is called *routing policies* configuration, and prevents BGP from using the shortest path. Moreover, it has been demonstrated that the Internet possesses rich path diversity at its core [8, 9], which BGP is not able to employ because it only selects one path. The second problem, called *persistent route oscillations* (PRO) [10], develops when a group of ASs cannot find a unique solution to a destination and oscillate between two or more of these. Ideas proposed to solve this problem try to prevent the development of oscillations by implementing an acyclic topological structure which bounds the policies an AS may define [11–16]. However, none of these considers the general topological structure of the Internet and most fail to exploit alternative paths to the same destination. A routing protocol which were able to exploit path diversity would not only increase the Internet's capacity, but also its resilience to failures and its stability. Conversely, it would also produce a complex system which may need to be bound to limit the overheads it will produce in the network's routers.

A new routing strategy, called *Chain Routing*, proposes to employ complete orders as a topological unit [7] that allows a group of ASs to safely define paths without causing PRO. This is because complete orders are acyclic digraphs that inherently posses more than one path to a destination. It is important to notice that this strategy is not intended to be a stand-alone routing protocol, but a framework that helps to perform this task.

The current paper analyses some of Chain Routing's features with an aim to mathematically demonstrate why complete orders are well-suited to abstract routing information at the interdomain level, to refine the definition of a mathematical concept, the Varc, which is needed to define complete orders, and to analyze the computational complexity of employing this algorithm.

This paper is organised as follows. Section 2 provides a mathematical analysis to show that complete orders can be employed to abstract the topological diversity of a network. Section 3 defines the Virtual Arc as a mathematical structure that is used to define complete orders in a network, while Section 4 gives further information on the practical application of Virtual Arcs and describes an experiment which demonstrates the feasibility of this task. Then, Section 5 discusses the main characteristics and limitations experienced by this routing framework.

2 Mathematical Background

In the specific case of BGP, paths are announced, adopted and inherited from one AS to its neighbours, so it becomes necessary to employ a *digraph*, $D = (V, A)$, where the *vertex set*, V, represents ASs, and the *arc set*, A, models how paths are learned and propagated in the Internet.

Path propagation in the Internet is restricted by the fact that BGP only selects one path and the policies that each AS implements. The *announcement digraph of ASi*, $D_{anc}(i)$ models how a network could propagate announcements if only policies restrict the announcement of destinations [8]. This digraph should have greater path diversity than the current network model. The converse of $D_{anc}(i)$ is

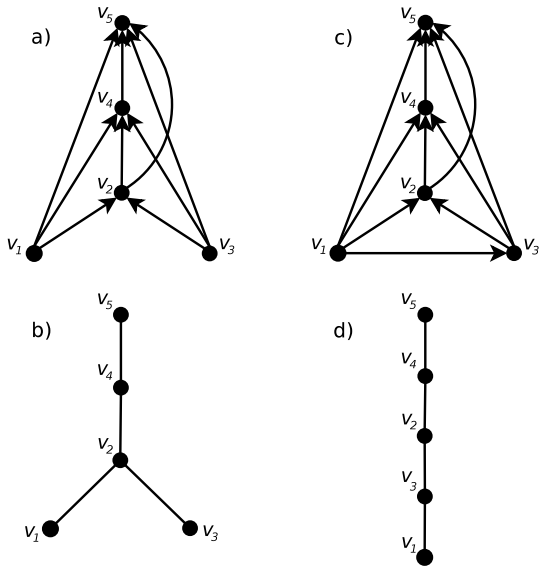

Fig. 1: Example of a partial order (a) with its Hasse diagram (b); and a complete order (c) with its Hasse diagram (d)

the *destination digraph of ASi*, $D_{dst}(i)$, and it represents how a group of ASs could send back information to a destination if BGP's restrictions do not apply.

Although $D_{anc}(i)$ and $D_{dst}(i)$ describe how destinations and information may propagate accross the Internet, they do not provide a metric to specify the connectivity of this network. The number of arc-disjoint paths has been proposed as a unit of path diversity between ASs [8]. *Arc-disjoint paths* is a set of directed paths from a source vertex to a destination vertex in which no common arcs are traversed more than once. This definition relates to *internally-disjoint paths*, in which neither the vertices nor the arcs can be traversed more than once.

The following definitions apply in this article: A *loop* is an arc with the same head and tail (v_1v_1). A *cycle* is a closed directed path formed by two or more arcs. An *acyclic digraph* is any digraph that has no cycles. A *partial order* [17] is an acyclic digraph which complies with these properties: *irreflexive* (there are no loops), *asymmetric* (if arc v_1v_2 exists, then arc v_2v_1 does not) and *transitive* (if arcs v_1v_2 and v_2v_3 exist, then arc v_1v_3 must exist). A partial order is said to be *complete*, if the previous three properties apply to *all* the vertices in the graph, and is then called a *chain* or a *complete order* [17, 18]. Fig. 1 shows examples of a partial order, a complete order and their corresponding Hasse diagrams.

Complete orders have properties that are useful to abstract topological information from a network and allow to use them as a routing framework [17]:

- All complete orders with n vertices have $n(n-1)/2$ arcs.
- The *height* (H) of a complete order is defined as the number of vertices minus one ($n-1$).
- Due to their completeness, all chains with the same n are *isomorphic*.
- Adding any new arc to a complete order will form a cycle, which means they are *maximal* acyclic digraphs.

In addition, complete orders have a unique *transmitter* or source vertex and a unique *receiver* or destination vertex. Which means that the maximal acyclic digraph, which is also the best connected one, between a source vertex and a destination is the complete order. However, there is no formal definition of how much path diversity a complete order provides between the source and destination vertices.

Theorem 1. *If C is a complete order with $n \geq 2$ vertices, then there are $n-1$ arc-disjoint (internally-disjoint) paths from the transmitter to the receiver.*

Proof: Following the induction method: the smallest possible chain has 2 vertices and 1 path (v_1v_2). Assume there is an n-vertex chain which has $n-1$ arc-disjoint (internally disjoint) paths from a transmitter v_1 to a receiver v_n. By employing the transitive property, the complete order with $n+1$ vertices also has $n-1$ arc-disjoint (internally disjoint) paths between v_1 and a new receiver v_{n+1}. In addition, arc v_1v_{n+1} implements a new arc-disjoint (internally disjoint) path, which has to exist due to completeness.

□

Theorem 1 demonstrates that the smallest chain which can offer path diversity needs 3 vertices and has 2 arc-disjoint paths. Notice that by definition $n-1$ is the chain's height (H).

Define u as the *number of arcs used by the $n-1$ arc-disjoint paths from the transmitter to the receiver*.

Theorem 2. *If C is a complete order with $n \geq 2$ vertices, then all of the $n-1$ arc-disjoint (internally disjoint) paths from the transmitter to the receiver use exactly $u = 2n-3$ arcs.*

Proof: Following the induction method: the smallest possible chain has 2 vertices and uses 1 arc for its only path. Assume there is an n-vertex chain which uses $2n-3$ arcs for its $n-1$ arc-disjoint (internally disjoint) paths from a transmitter v_1 to a receiver v_n. This set of arcs must implement 1 direct path from the transmitter to the receiver (v_1v_n) and $n-2$ paths from v_1 to an intermediate node v_i and then to v_n. The complete order with $n+1$ vertices must have a new arc v_nv_{n+1} and, because of completeness, it also must have the arc v_1v_{n+1}. Moreover, transitivity determines that if there is an arc between any intermediate vertex v_i and v_n and v_nv_{n+1} exists, then all arcs v_iv_{n+1} also exist and they also employ $2n-3$ arcs for their arc-disjoint (internally-disjoint) paths between v_1 and v_{n+1}. Then, the total number of arcs used u is:

$$u = 2n-3+1+1 = 2(n+1)-3. \quad (1)$$

□

The *number of remaining or unused arcs* (r) is found by subtracting u in Theorem 2 from the chain's total number of arcs:

Corollary 3. *If C is a complete order with $n \geq 2$ vertices, there are exactly $r = (n-2)(n-3)/2$ arcs that are not used in all the $n-1$ arc-disjoint (internally disjoint) paths from the transmitter to the receiver.*

Theorem 2 and corollary 3 show that r increases quadratically with respect to n, while u increases linearly. For the smallest $n = 2$ chain, $u = 1$ and $r = 0$. However, when $n = 8$, then $u = 13$ and $r = 15$. Therefore, if $n > 7$, then $r > u$ follows. This means that when the number of nodes in the chain reaches 8, routers will need to spend more memory and computational resources with path's coordination tasks which do not belong to the efficient arc-disjoint paths. Computational resources are further discussed in Section 5.1.

Furthermore, it has been demonstrated that, when a chain C of n vertices eliminates 1 vertex ($C-v$), the resulting graph is the complete order of $n-1$ vertices [17]. The following theorem analyses the inverse operation: adding 1 vertex to a chain and thus increasing H by 1.

Theorem 4. *If C is a complete order with $n \geq 2$ vertices, then the augmented complete order $C+1$ which has exactly 1 more vertex than C requires n new arcs.*

The previous theorem is proven by subtracting the total number of arcs in $C+1$ minus the same number for C.

Corollary 5. *Increasing the number of vertices and height of a complete order by 1 requires at least 2 new arcs.*

In summary, the acyclic digraph with the greatest number of arc-disjoint paths between a source and a destination is the complete order. These two properties had led to the conclusion that complete orders could be used to abstract topological information from a network with directed paths. The following section shows how these concepts could be applied to define paths in a BGP network.

3 Defining the Virtual Arc

Complete orders may not be formed directly from an ASs interdomain network. Therefore, it becomes necessary to define a structure which allows to map this type of orders between ASs in the Internet. The structure employed to perform this mapping is called a Virtual Arc [7], and it allows to abstract a series of arcs that follow a path between two of the chain's vertices into a single arc. This mathematical structure is defined as:

Definition 1. *A* Varc *or* Virtual Arc *is a structure that represents a set of vertices and their adjacent arcs that follow a directed path from an initial vertex x to a final vertex y, where the directed path needs to be abstracted in order to allow x and y to be the end vertices of a chain segment and neither of the intermediate vertices is also included in the chain.*

The Varc allows to map a sequence of ASs into a complete order. This sequence cannot be regarded as part of the complete order, but it allows to transmit packets through the ASs that form it. An example of how to use Varcs to define complete orders in a network is depicted in Fig. 2, where the main chain, C_1, is formed by vertices v_1, v_7, v_3 and v_4. This figure also shows that two segments in this chain have been abstracted as Varcs: v_1v_3 and v_1v_4, while segment v_1v_7 has been abstracted as chain C_2. Both abstractions hide connectivity information that cannot be stored at the initial chain. This recursivity implements an abstraction of the network's topological structure which, in turn, allows to discover which ASs are needed to implement a routing coordination strategy that could increase path diversity and improve resilience in the network. Due to the way Varcs can be discovered when initially exploring the network, two different types have been defined:

- *Simple Varcs* are formed by a series of vertices and their adjacent arcs where each intermediate vertex v_i has only one arc that enters each v_i and one arc that leaves each v_i. Simple Varcs may exist independently without any need to be included in a chain and they can be used as regular arcs. Both Varcs in Fig. 2 are simple.
- *Functional Varcs* are also formed by a series of vertices and their adjacent arcs, but there is no restriction on the number of arcs that enter and leave any of the intermediate vertices.

Simple Varcs are easily found when new arcs are included in the data structure, but Functional Varcs can only be defined when needed, otherwise an extremely large number of Functional Varcs could be found in a network. Therefore, each type of Varc needs to be defined at different stages of the algorithm. Also, Functional Varcs could be defined differently for different chains. For example, Fig. 3 shows how FVarc(v_1v_3) is employed at C_3 and Fvarc(v_2v_4) at C_4. Notice that these two Varcs share a common arc.

4 Implementing Virtual Arcs

An early implementation of the Chain Routing strategy used a modified BFS algorithm to prove the feasibility of employing this structure [7]. Although this early program provided positive results, it was not able to complete the processing for the network employed. Section 4.1 introduces new development towards achieving this objective, while Section 4.2 describes an experiment that allows to analyse the results.

4.1 Chain Routing algorithm

Chain Routing's first implementation [7] was based in the Breadth-First Search (BFS) algorithm [18] because this algorithm first finds the direct paths from the source to the destination, and then discovers paths that employ intermediate (transitive) nodes. This implementation, which was named the *Modified BFS algorithm*, is still used with some modifications (Algorithm 1).

The input to the Modified BFS algorithm is an announcement digraph's $D_{anc}(i)$ adjacency matrix, and its objective is to build a data structure that abstracts the topological information of such graph. Vertices which are being discovered ($distance(y) = nil$) are added to the data structure as either an arc, or as the last arc of a Simple Varc, while vertices that have already been discovered should be included in a chain. Further processing is required to determine the chains that may be defined with this new arc. A procedure called *add_arc_to_Chain* (Algorithm 2) has been developed to perform this task. Importantly, Functional Varcs are defined at the *add_arc_to_chain* procedure only after the other types of structures have failed to define a chain. This gives priority to the definition of higher chains and chains formed by simpler structures, like arcs or Simple Varcs. Notice how Simple Varcs are instead defined at Algorithm 1.

Before analysing procedure *add_arc_to_Chain*, it is necessary to identify two different types of arcs: The 3-vertex chain shown in Fig.

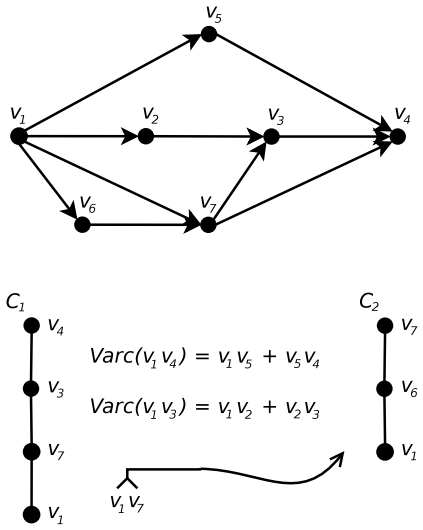

Fig. 2: Chain Routing Data Structure example

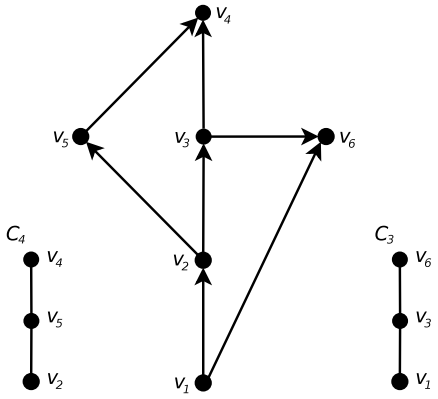

Fig. 3: Functional Varc example

Algorithm 1: Modified BFS algorithm

Input: An adjacency matrix which represents $D_{anc}(i) = (V, A)$

foreach $v \in V$ **do**
  predecessor(v) ← nil;
  distance(v) ← nil;
end

distance(i) ← 0; `// Where i is the source vertex`
QUEUE ← i;

while *QUEUE* $\neq \emptyset$ **do**
  x ← head of QUEUE (simultaneously remove x from QUEUE);
  foreach *out-neighbour (y) of* x **do**
    if *distance(y) = nil* **then**
      distance(y) ← distance(x) + 1;
      predecessor(y) ← x;
      tail of QUEUE ← y;
      if *(y is the only out-neighbour of x) AND ($x \neq i$)* **then**
        A **Simple Varc** is included into the data structure using arc xy and the predecessor structure of x;
      else
        Arc xy is included into the data structure;
      end
    else
      if *predecessor(y) = nil* **then**
        Ignore Arc xy because it forms a cycle;
      else
        Use (transitive) arc xy to include a **Chain** into the data structure;
      end
    end
  end
end

Algorithm 2: *add_arc_to_Chain*

Input: The start and end vertices of a transitive arc: $x_n y_n$

x_{n-1} ← predecessor(x_n);
y_{n-1} ← predecessor(y_n); `// Usually, x_{n-1} = y_{n-1},`
`// which is also the chain's source vertex`
X_num ← number of occurrences of $x_{n-1}x_n$ in the database;
Y_num ← number of occurrences of $y_{n-1}y_n$ in the database;

foreach *Y_num* **do**
  Y_level ← find the level of arc $y_{n-1}y_n$ in the database;
  foreach *X_num* **do**
    X_level ← query Arc db for level of arc $x_{n-1}x_n$;
    X_type ← Structure to which $x_{n-1}x_n$ belongs in the database;
    switch *X_type* **do**
      case *Arc* **do**
        $x_{n-1}x_n$ is an **Arc** and could be used to form a Chain;
      end
      case *Chain* **do**
        $x_{n-1}x_n$ is a **Chain** and may combine with other Chains to form longer chains. If not, it could be used to form a Chain;
      end
      case *Varc* **do**
        $x_{n-1}x_n$ is a **Simple Varc** and could be used to form a Chain;
      end
      case *FVarc* **do**
        $x_{n-1}x_n$ is a **Functional Varc** and could be used to form a Chain;
      end
    end
  end
end

4 has source vertex x_{n-1}, and destination vertex y_n. A *transitive arc*, $x_n y_n$, allows to reach y_n through two other arcs: $x_{n-1}x_n$ and $x_{n-1}y_n$. These are called the *predecessor arcs* of $x_n y_n$.

Procedure *add_arc_to_Chain* receives a transitive arc and its predecessor arcs as input. The two predecessor arcs may not have a common source vertex, therefore they are initially labelled as $x_{n-1}x_n$ and $y_{n-1}y_n$. If $x_{n-1} = y_{n-1}$, then x_{n-1} is the source of a 3-vertex chain; but if $x_{n-1} \neq y_{n-1}$, further processing is required to determine the best strategy to form a new chain.

Each predecessor arc of a newly discovered transitive arc may have been included before in other structures, or they could be single arcs in the data structure. Corollary 5 demonstrated that more than one arc is needed to increase the chain's height. However, a single arc may allow to combine two chains to define a higher chain. If neither of the predecessor arcs does not belong to a chain, the best possible scenario is to define a 3-vertex chain.

Since each predecessor arc may already be part of other structures, more than one occurrence of these arcs could exist in the data structure. Therefore it is necessary to test each occurrence until a best scenario is found: a chain with $H > 2$. Algorithm 2 shows how

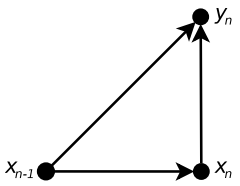

Fig. 4: The transitive arc of a Chain

predecessor arcs are processed using two nested loops. The loops try to find which structure each predecessor arc belongs to, starting by arc $x_{n-1}x_n$. The switch structures that test arc $y_{n-1}y_n$ in each section of the $x_{n-1}x_n$ switch are too long to be included, but they are similar to the structure already displayed. Algorithm 2 makes a distinction between Simple and Functional Varcs because Simple Varcs are used just as an Arc, while Functional Varcs need to be processed to find a suitable configuration for the chain.

4.2 Testing the Chain Routing algorithm

The early implementation of the Chain Routing algorithm was tested using 45 announcement digraphs obtained by analysing the topology from the main European ASs [8]. This implementation had many unfinished parts that prevented the program to complete the processing for all 45 graphs [7]. The current implementation has been finished to the point where it can process all 45 original announcement digraphs. More importantly, the current algorithms have been modified to make a clear distinction when processing and handling Simple and Functional Varcs, which provides significant enhancements when trying to construct the data structure that abstracts the network topology.

In order to verify these improvements, the 45 original announcement digraphs [8] were processed again. Then, the highest chain from each originating AS to the others was recorded. The most common occurrence is the 3-vertex chain ($H = 2$). If there is only one path from the source vertex to another vertex, but this path is also part of a longer chain, then the vertex is recorded to have a chain of height 1; however, if the only path to a vertex is not part of a longer chain, then the vertex is reached through an arc (A). A third definition is that of a bridge (B), which is used when some form of path diversity is constrained by a single arc. These results were

Table 1 Current implementation's results for highest possible chain or structure

AS	1299	702	3303	1257	13237	8220	286	3257	1273	16150
1299	-	1	3	A	A	A	2	A	2	B
702	2	-	2	2	3	3	3	3	2	**4**
3303	2	1	-	2	2	4	2	2	2	**2**
1257	2	1	2	-	2	A	2	A	2	B
13237	2	2	2	2	-	2	3	2	2	2
8220	2	1	2	A	2	-	2	1	2	2
286	2	1	2	A	2	2	-	A	1	2
3257	2	2	2	2	2	2	3	-	2	2
1273	2	2	2	2	2	2	3	2	-	2
16150	**2**	2	3	**2**	2	3	3	2	**1**	-

recorded in a large (45×45) table which has been included at the Appendix (Tables 4 to 6).

For convenience to the reader, Table 1 shows the highest recorded chain for the 10 largest European ASs. These results may be compared with Table 1 in [7]. Closer inspection of both tables uncovers that the majority of results has remained and only 5 recordings have changed (in bold). Four recordings have increased and only one has decreased.

Fig. 5 displays the count of all heights in Tables 4 to 6. To verify the performance of the current implementation, Table 2 shows the percentage of each chain-height frequency for the first [7] and second implementations. $H = 1$ is the only decreasing percentage frequency and, as an absolute value, $H = 4$ is the one that has increased the most.

The current implementation of the Chain Routing framework is able to process the original 45 announcement digraphs of the European ASs' network core, and to abstract network topological information to the point where it can define chains with $H = 4$. Therefore, this algorithms could be employed as an extension to BGP in which ASs included in Varcs are abstracted inside a chain structure. The following section explores other important properties inherent to the Chain Routing algorithm.

Table 2 Chain-height (H) frequency count and as a percentage

Chain-height (H)	1	2	3	4
Count 1st implem.	119	670	71	2
Perc. 1st implem.	13.8%	77.7%	8.2%	0.2%
Count 2nd implem.	128	999	113	15
Perc. 2nd implem.	10.2%	79.6%	9.0%	1.2%

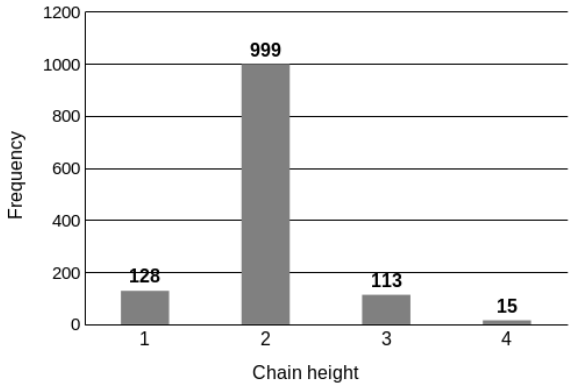

Fig. 5: Chain-height (H) frequency count

5 Discussion

5.1 Analysing the Chain Routing algorithm

The Chain Routing algorithm is NP-complete and this can be proved by reducing a variation of the scheduling algorithm [19, 20].

Instance: A directed graph $D = (V, A)$ (not necessarily acyclic) with an origin vertex, v_0, and a spanning tree that originates at v_0 and includes the arc set $(A_{span} \subseteq A)$. A partial order on V. A variable number, C, of complete orders or chains and an integer, H_{max}, which represents the maximum height of all complete orders.
Question: Is it possible to map V to C different complete orders?

Although Varcs are not explicitly defined at the Instance, the fact that they are always further away from the origin vertex v_0 implies that the order is not broken, hence the use of a partial order at the Instance of this algorithm. Also notice that C and H_{max} are variables and that the best scenario would be where C has the smallest possible value and H_{max} the largest.

Theorem 6. *The Chain Routing algorithm is NP-Complete.*

Proof: This will be done by reduction of the precedence constrained scheduling problem [19, 20]. Specifically, this problem defines m number of machines (or processors), a precedence relation, a deadline or completion time T and every job requires exactly one time unit. Since scheduling operates in the time domain and time is a complete order, there is a clear one-to-one analogy to the chain routing algorithm. There is also an analogy between the number of complete orders C found in D and the m used to process each job. The main difference between these two problems is that while the scheduling problem tries to find the minimum completion time T, chain routing tries to establish the largest value for H_{max}. It is easy to see that in order to find either T or H_{max} it is necessary to follow the same procedure and because all jobs require one time unit, it is possible to map each job in the scheduling problem to a vertex in the chain routing problem.

□

Although Theorem 6 demonstrates that the proposed algorithm is NP-Complete, Theorem 2 and Corollary 3 have also proved that it is not efficient to use chains with $H > 7$. Therefore, practical limitations to the proposed algorithm also set a bound in the computational steps needed to abstract a destination digraph using the Chain Routing framework.

5.2 Related Work

Chain Routing is neither the only nor the first proposal to increase the Internet's path diversity. Some proposals have been offered to enhance BGP's capabilities. Special attention has been given to Neighbour-Specific BGP (NS-BGP) [21] which proposes to extend BGP to allow ASs to announce different paths to different neighbours wanting to reach the same destination. This is possible because

of the Internet's rich path diversity and because it is not possible to define a best route that fits the needs of every AS. By increasing path selection flexibility, NS-BGP implements a routing system which, in theory, takes advantage of the Internet's path diversity and may be safer than traditional BGP. However, this idea needs further testing to provide a clear understanding of its capabilities and limitations.

Other proposals have tried to employ alternative paths to destinations at the expense of producing scalability issues. In order to control these, they have either hidden topological information [22] or implemented other mechanisms, e.g.: a Bloom Filter [23].

The Simple Path Diversity Algorithm (SPDA) proposed in [24] is able to provide alternative paths to a destination. Unfortunately, the authors admit that this strategy cannot prevent the development of cycles in the paths. Conversely, other studies have focused on avoiding network instability, but not on exploiting alternative paths [25, 26].

However, all these previous solutions are basically trying to enhance BGP's normal functionality to support a multiple path scheme, while our approach is to employ an acyclic mathematical structure to find the *maximum path diversity* in a topology and exploit this resource to perform routing without causing instabilities. Moreover, when a group of ASs is coordinated through a complete order, the failure of a node or a link does not require the network to re-converge, as the inherent path diversity means that routing can continue uninterrupted until a convenient time to recalculate the routing topological structure is decided upon.

6 Conclusions

This paper has presented the mathematical foundation used to develop a new routing framework, Chain Routing, which employs complete orders as its main topological unit. The advantages of employing complete orders are: they provide a defined path diversity that should support the implementation of traffic engineering; they implement a structure that allows ASs to find alternative paths when a node or a link fails. Moreover, complete orders avoid the development of transient loops and persistent route oscillations because they are acyclic.

The paper also introduces the Virtual Arc mathematical concept which is used to implement Chain Routing in a real network. An experiment using a real topology proves that it is possible and convenient to employ Varcs and chains to model and abstract the topological information of a digraph. Furthermore, the analysis in this article demonstrates that Chain Routing tries to solve an NP-Complete problem. Fortunately, it is still possible to employ this algorithm to increase the network's path diversity by implementing chains with limited height, thus setting a bound to the number of computational tasks that need to be performed.

Although Chain Routing may allow the exploitation of the Internet's path diversity which BGP cannot take full advantage of, there are some important factors that still need to be considered before this routing framework is implemented. For example, Chain Routing increases the message overhead needed to exchange topological information. Also, route aggregation and topological adjustments will require further processing to modify the orders already defined. However, this may be a small price to pay for a more stable and resilient routing strategy.

Chain Routing has the potential to improve the routing functionality provided by BGP, but before a practical routing protocol can be developed, a large amount of experimentation is needed. For this reason it is important to continue the research on algorithms that would enable the implementation of this new routing strategy.

7 Acknowledgements

This research was in part supported by Mexico's Conacyt and Prodep.

8 References

1 Rekhter, Y., Li, T., Hares, S.: RFC 4271: 'A Border Gateway Protocol 4 (BGP-4)' (Internet Society Std.), 2006
2 Labovitz, C., Ahuja, A., Bose, A., Jahanian, F.: 'Delayed Internet routing convergence'. Proc. ACM SIGCOMM, Stockholm, Sweden, 3, September 2000, pp. 175-187
3 Labovitz, C., Ahuja, A., Jahanian, F.: 'Experimental study of Internet stability and backbone failures'. 29th Int. Symposium on Fault-Tolerant Computing, June 1999, pp. 278-285
4 Mao, Z.M., Govindan, R., Varghese, G., Katz, R.H.: 'Route flap damping exacerbates Internet routing convergence'. Proc. ACM SIGCOMM, Pittsburgh, Pennsylvania, August 2002, pp. 221-233
5 Chandrashekar, J., Duan, Z., Zhang, Z., Krasky, J.: 'Limiting path exploration in BGP'. Proc. IEEE INFOCOM, 4, March 2005, pp. 2337-2348
6 Pei, D., Azuma, M., Massey, D., Zhang, L.: 'BGP-RCN: improving BGP convergence through root cause notification', Computer Networks, 2005, 48, (2), pp. 175-194
7 Arjona-Villicaña, P.D., Constantinou, C.C., Stepanenko, A.S.: 'Chain Routing: a new routing framework for the Internet based on complete orders', IET Communications, 2011, 5, (16), pp. 2345-2355
8 Arjona-Villicaña, P.D., Constantinou, C.C., Stepanenko, A.S.: 'The Internet's unexploited path diversity', IEEE Communications Letters, 2010, 14, (5), pp. 474-476
9 Oliveira, R., Pei, D., Willinger, W., Zhang, B., Zhang, L.: 'In search of the elusive ground truth: the Internet's AS-level connectivity structure'. Proc. ACM SIGMETRICS, June 2008, pp. 217-228
10 Varadhan, K., Govindan, R., Estrin, D.: 'Persistent route oscillations in inter-domain routing', Computer Networks, 2000, 31, (1), pp. 1-16
11 Cittadini, L., Di Battista, G., Rimondini, M.: '(Un)-stable routing in the Internet: a survey from the algorithmic perspective', Graph-Theoretic Concepts in Computer Science: WG 2008, LNCS, 5344, 2008, pp. 1-13
12 Cobb, J.A., Musunuri, R.: 'Enforcing convergence in inter-domain routing': Proc. IEEE GLOBECOM, 3, November 2004, pp. 1353-1358
13 Ee, C.T., Chun, B.G., Lakshminarayanan, K., Ramachandran, V., Shenker, S.: 'Resolving inter-domain policy disputes'. Proc. ACM SIGCOMM, Kyoto, Japan, August 2007, pp. 157-168
14 Gao, L., Rexford, J.: 'Stable Internet routing without global coordination'. Proc. ACM SIGMETRICS, 6, June 2000, pp. 307-317
15 Griffin, T.G., Shepherd, F.B., Wilfong, G.: 'The stable paths problem and inter-domain routing', IEEE/ACM Transactions on Networking, 2002, 10, (2), pp. 232-243
16 Griffin, T.G., Shepherd, F.B., Wilfong, G.: 'Policy disputes in path-vector protocols'. Proc. 7th Int. Conf. on Network Protocols (ICNP), November 1999, pp. 21-30
17 Harary, F., Norman, R.Z., Cartwritght, D.: 'Structural models: An introduction to the theory of directed graphs', J. Wiley & Sons, 1965
18 Bang-Jensen, J., Gutin, G.: 'Digraphs: Theory, algorithms and applications', Springer, 2002
19 Ullman, J.D.: 'NP-Complete Scheduling Problems', Journal of Computer and System Sciences, 1975, 10, pp. 384-393
20 Lenstra, J.K., Rinnooy Kan, A.H.G.: 'Complexity of Scheduling under Precedence Constraints', Operations Research, 1978, 26, (1), pp. 22-35
21 Wang, Y., Schapira, M., Rexford, J.: 'Neighbor-Specific BGP: more flexible routing policies while improving global stability'. Proc. ACM SIGMETRICS, June 2009, pp. 217-228
22 Ganichev, I., Dai, B., Godfrey, P.B., Shenker, S.: 'YAMR: yet another multipath routing protocol', ACM SIGCOMM Computer Communications Review, 2010, 40, (5), 13-19
23 Wang, F., Gao, L.: 'Path Diversity Aware Interdomain Routing'. Proc. IEEE INFOCOM, April 2009, pp. 307-315
24 Shieh Y.P., Hsu, W.H.: 'Simple path diversity algorithm for interdomain routing', IET Communications, 2011, 5, (16), pp. 2310-2316
25 Agarwal, R., Jalaparti, V., Caesar, M., Godfrey, P.B.: 'Guaranteeing BGP Stability with a Few Extra Paths'. Proc. IEEE ICDCS, June 2010, pp. 221-230
26 Kushman, N., Kandula, S., Katabi, D., Maggs, B.M.: 'R-BGP: staying connected in a connected world'. 4th USENIX NSDI, April 2007

9 Appendix

Tables 4 to 6 show the highest chain or other structure employed to connect a source AS (column) to a destination AS (row). These results are meant to be in a single table, but they had to be divided because of space. Also, in order to minimise space, the number of the ASs are not displayed, instead there is a code that starts in S1 and finishes in S45. The AS number that corresponds to each code is shown at the following Table 3:

Table 3 Code for each AS number included in Tables 4 to 6

Code	AS number	Code	AS number
S1	AS1299	S24	AS3301
S2	AS702	S25	AS8434
S3	AS3303	S26	AS5089
S4	AS1257	S27	AS6878
S5	AS13237	S28	AS31399
S6	AS8220	S29	AS6667
S7	AS286	S30	AS1103
S8	AS3257	S31	AS2603
S9	AS1273	S32	AS680
S10	AS16150	S33	AS3269
S11	AS8928	S34	AS3215
S12	AS8342	S35	AS9121
S13	AS5413	S36	AS786
S14	AS5511	S37	AS8404
S15	AS12956	S38	AS20485
S16	AS6762	S39	AS9035
S17	AS15412	S40	AS1267
S18	AS5400	S41	AS3352
S19	AS20965	S42	AS6830
S20	AS3292	S43	AS3216
S21	AS3246	S44	AS5568
S22	AS6805	S45	AS3356
S23	AS8210		

Table 4 Highest chain or other structure from a source AS to a destination AS (Part 1)

AS	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
S1	-	1	3	A	A	A	2	A	2	B	2	2	2	1	2
S2	2	-	2	2	3	3	3	3	2	4	3	2	2	2	2
S3	2	1	-	2	2	4	2	2	2	2	3	3	3	3	2
S4	2	1	2	-	2	A	2	A	2	B	2	2	3	1	2
S5	2	2	2	2	-	2	3	2	2	2	2	2	2	B	2
S6	2	1	2	A	2	-	2	1	2	2	2	2	3	2	1
S7	2	1	2	A	2	2	-	A	1	2	2	2	3	0	2
S8	2	2	2	2	2	2	3	-	2	2	2	2	2	2	2
S9	2	2	2	2	2	2	3	2	-	2	2	3	2	2	2
S10	2	2	3	2	2	3	3	2	1	-	2	3	2	2	2
S11	3	4	2	2	2	2	3	2	2	2	-	3	2	2	2
S12	3	2	2	B	2	2	2	2	2	2	3	-	2	2	2
S13	2	2	2	2	2	2	3	2	2	2	2	3	-	1	2
S14	2	1	2	A	B	B	2	A	2	2	2	2	2	-	2
S15	2	3	3	2	B	2	3	2	2	B	2	2	2	2	-
S16	2	1	2	A	B	0	2	A	2	B	2	2	2	B	2
S17	1	1	3	2	2	2	2	2	2	2	2	2	2	2	2
S18	2	1	2	A	B	2	2	A	2	2	2	2	3	1	2
S19	1	2	2	B	B	B	B	2	2	B	2	2	B	2	2
S20	2	2	3	2	2	2	3	2	2	3	2	2	3	B	2
S21	1	2	2	2	2	B	2	2	2	2	2	2	2	B	2
S22	2	2	3	B	2	A	2	2	2	2	2	2	2	2	2
S23	0	1	2	A	A	0	A	A	1	2	A	2	2	0	B
S24	1	B	B	2	B	B	B	B	B	2	B	B	B	B	B
S25	2	2	2	B	2	B	B	2	1	2	2	2	B	2	2
S26	3	4	4	3	2	2	4	3	2	3	2	2	2	2	2
S27	2	2	3	2	2	2	2	2	2	2	2	B	2	B	2
S28	2	2	2	2	2	2	3	2	1	2	2	2	2	2	2
S29	2	2	3	2	1	2	3	2	2	2	2	2	2	2	2
S30	2	1	2	2	2	B	2	2	1	2	2	2	2	B	2
S31	2	2	2	2	2	B	2	B	1	2	2	3	2	B	B
S32	2	2	2	B	2	3	2	2	1	2	2	3	2	B	B
S33	2	2	2	B	B	A	2	B	1	B	2	2	B	B	2
S34	B	B	B	B	B	2	B	B	B	B	1	B	B	1	B
S35	2	4	4	2	B	2	2	3	3	2	2	3	B	2	2
S36	2	2	2	2	2	2	3	2	2	2	2	2	2	2	B
S37	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
S38	2	3	3	2	3	2	3	2	2	2	2	3	2	2	2
S39	3	2	3	2	B	B	2	2	2	B	2	2	2	2	2
S40	3	2	3	2	2	B	2	3	2	B	2	2	2	2	2
S41	B	B	B	B	B	2	B	B	B	2	2	B	B	B	1
S42	3	3	4	3	2	2	3	2	2	2	2	2	2	3	3
S43	3	2	3	2	2	2	2	2	2	2	2	3	2	2	2
S44	3	4	2	2	3	3	3	2	2	2	2	2	2	2	2
S45	2	1	A	0	2	2	2	A	1	2	2	2	2	1	1

Table 5 Highest chain or other structure from a source AS to a destination AS (Part 2)

AS	S16	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30
S1	1	B	A	2	2	B	2	B	A	B	A	B	B	2	B
S2	2	B	3	2	2	3	3	B	B	2	2	B	A	3	2
S3	2	2	2	2	2	2	3	2	B	2	2	B	B	2	2
S4	1	2	A	2	2	2	2	2	2	B	0	B	B	3	2
S5	B	2	B	2	2	2	2	B	B	2	2	B	B	2	2
S6	2	2	2	2	2	2	2	B	B	B	2	2	B	3	B
S7	2	2	A	2	2	2	3	2	B	B	2	B	B	2	2
S8	2	2	2	2	2	2	2	2	B	1	2	B	B	3	2
S9	2	2	2	2	2	2	2	B	B	2	2	B	B	2	A
S10	2	2	2	2	2	3	2	2	2	3	2	B	B	2	2
S11	2	2	2	2	2	2	2	1	B	2	2	B	B	2	B
S12	2	2	2	2	2	2	2	2	B	2	2	B	B	2	B
S13	2	2	2	B	2	B	2	B	B	1	2	B	B	2	A
S14	1	B	A	2	2	B	2	B	B	2	2	B	B	3	B
S15	2	B	2	2	4	B	2	B	B	B	2	B	B	2	B
S16	-	2	A	2	2	B	2	B	B	B	0	B	B	2	B
S17	2	-	2	2	2	3	2	2	B	2	2	B	B	2	A
S18	1	2	-	2	2	B	2	2	B	2	B	B	B	2	B
S19	2	B	2	-	2	B	2	B	B	B	B	B	B	2	A
S20	2	2	2	2	-	2	3	2	2	2	2	B	B	2	2
S21	1	2	B	2	2	-	2	2	2	2	2	B	B	3	2
S22	2	2	2	2	2	2	-	2	B	1	2	A	B	2	A
S23	0	2	A	B	2	2	2	-	0	1	A	B	B	2	A
S24	B	B	B	B	B	2	B	1	-	2	B	B	B	B	B
S25	2	2	2	2	3	2	2	2	2	-	B	B	B	2	2
S26	2	2	2	2	2	2	3	2	B	2	-	B	B	2	2
S27	1	2	2	B	2	2	2	2	B	1	2	-	B	2	2
S28	2	2	2	2	2	2	2	2	B	B	2	1	-	2	2
S29	2	2	B	2	3	2	2	2	B	2	A	B	B	-	A
S30	B	2	B	2	2	2	2	2	B	1	0	B	B	2	-
S31	B	B	B	2	3	2	B	1	2	2	B	B	B	3	2
S32	2	B	2	2	2	B	2	B	B	B	2	B	B	2	2
S33	1	B	B	2	2	B	B	B	B	B	0	B	B	2	B
S34	B	B	B	B	B	B	B	B	B	B	0	B	B	B	B
S35	4	3	2	2	2	B	2	2	B	2	2	B	B	3	2
S36	B	2	2	2	2	2	B	2	B	2	2	B	B	2	2
S37	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
S38	2	2	2	B	2	2	2	2	B	2	2	B	B	3	2
S39	2	B	2	2	2	B	2	B	B	B	0	B	B	2	B
S40	2	B	2	2	2	B	B	B	B	B	2	B	B	2	B
S41	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
S42	2	2	2	2	2	B	2	2	B	2	2	B	B	2	2
S43	2	2	2	2	2	2	2	2	B	2	2	B	B	2	2
S44	2	2	2	2	2	2	2	3	B	2	2	B	B	2	2
S45	1	A	A	2	2	2	2	1	B	2	2	B	B	2	B

Table 6 Highest chain or other structure from a source AS to a destination AS (Part 3)

AS	S31	S32	S33	S34	S35	S36	S37	S38	S39	S40	S41	S42	S43	S44	S45
S1	2	A	B	B	2	A	B	2	2	B	B	2	2	B	1
S2	A	2	2	B	B	2	B	2	2	B	B	3	2	B	2
S3	B	2	2	B	B	2	B	2	2	B	B	2	2	B	3
S4	2	B	B	B	B	2	B	2	2	2	B	2	2	B	1
S5	2	2	B	B	B	2	B	2	2	B	B	2	3	B	2
S6	2	B	2	2	B	2	B	2	2	B	2	2	2	B	2
S7	B	2	B	0	B	2	B	2	2	B	B	2	2	B	B
S8	B	2	2	B	B	B	B	2	2	2	2	2	2	B	1
S9	B	B	B	B	A	2	B	2	2	B	B	2	2	A	1
S10	2	2	B	B	B	2	B	2	2	B	B	2	2	B	B
S11	2	2	2	B	B	B	B	2	2	B	2	2	2	B	2
S12	B	B	B	B	B	2	B	2	2	B	B	2	2	B	2
S13	B	B	B	B	B	2	B	2	2	B	B	2	3	B	1
S14	2	B	B	A	B	2	B	2	2	B	B	2	2	B	B
S15	2	B	B	B	B	2	B	2	2	B	A	2	2	B	2
S16	2	B	A	B	2	B	B	2	3	B	B	2	2	B	1
S17	B	B	B	B	B	2	B	2	2	B	B	2	2	B	2
S18	B	2	B	B	B	2	B	2	2	B	2	2	2	B	B
S19	A	2	B	B	B	2	B	B	2	B	B	2	2	B	2
S20	2	2	B	B	B	2	B	2	2	B	B	2	2	B	2
S21	2	2	B	B	B	2	B	2	2	B	B	2	3	B	2
S22	B	2	2	B	B	B	B	2	2	B	B	2	2	B	2
S23	A	0	0	0	B	A	B	2	2	B	B	2	2	B	1
S24	2	B	B	B	B	B	B	B	B	B	B	B	B	B	1
S25	2	B	B	B	B	B	B	2	2	B	B	2	2	B	1
S26	B	B	B	B	B	2	B	2	2	B	B	3	2	B	2
S27	B	2	2	B	B	B	B	2	2	B	2	2	B	B	2
S28	B	2	B	B	B	B	B	2	2	B	B	2	2	B	2
S29	2	B	B	B	B	2	B	2	3	B	B	2	2	B	3
S30	2	2	B	B	B	2	B	2	2	B	B	1	2	B	2
S31	-	2	B	B	B	2	B	2	B	B	B	B	2	B	2
S32	2	-	B	B	B	2	B	2	B	B	B	2	2	B	2
S33	B	B	-	B	B	B	B	2	2	2	B	2	2	B	1
S34	B	B	B	-	B	B	B	B	B	B	B	B	B	B	B
S35	B	2	B	B	-	B	B	2	2	B	B	2	2	B	4
S36	2	2	B	B	B	-	B	2	B	B	B	2	3	B	2
S37	B	B	B	B	B	B	-	B	B	B	B	A	B	B	B
S38	B	2	B	B	B	2	B	-	2	B	2	2	4	2	2
S39	B	B	2	B	B	B	B	2	-	B	B	2	2	B	4
S40	2	B	2	B	B	2	B	2	2	-	B	2	2	B	2
S41	B	B	B	B	B	B	B	B	B	B	-	B	B	B	B
S42	B	2	B	B	B	1	A	2	2	B	B	-	2	B	2
S43	B	B	B	B	B	2	B	2	2	B	B	2	-	2	3
S44	1	B	B	B	B	2	B	3	2	B	B	2	2	-	3
S45	2	2	B	B	2	2	B	2	2	B	2	2	2	B	-